Cours 2

Introduction

D) Un exemple…

- Pile abstraite et diverses implémentations

Type abstrait de données

NOM
  pile[X]
FONCTIONS
  vide : pile[X] -> Boolean
  nouvelle : -> pile[X]
  empiler : X x pile[X] -> pile[X]
  dépiler : pile[X] -> X x pile[X]
PRECONDITIONS
  dépiler(s: pile[X]) <=> (not vide(s))
AXIOMES
  forall x in X, s in pile[X]
    vide(nouvelle())
    not vide(empiler(x,s))
    dépiler(empiler(x,s))=(x,s)

Remarques

- Le type est paramétré par un autre type
- Les axiomes correspondent aux pré conditions
- Il n'y pas de représentation
- Il faudrait vérifier que cette définition caractérise bien un pile au sens usuel du terme (c'est possible)

Pile abstraite en java

```
package pile;


abstract class Pile <T>{
    abstract public T empiler(T v);
    abstract public T dépiler();
    abstract public Boolean estVide();
}
```

Divers

- `package`: regroupement de diverses classes
- `abstract`: signifie qu'il n'y a pas d'implémentation
- `public`: accessible de l'extérieur
- La classe est paramétrée par un type (java 1.5)

Implémentations

- On va implémenter la pile:
  - avec un objet de classe `Vector` (classe définie dans `java.util.package`) en fait il s'agit d'un `ArrayList`
  - Avec un objet de classe `LinkedList`
  - Avec `Integer` pour obtenir une pile de `Integer`

Une implémentation

```
package pile;
import java.util.EmptyStackException;
import java.util.Vector;
public class MaPile<T> extends Pile<T>{
    private Vector<T> items;
    // Vector devrait être remplacé par ArrayList
    public MaPile() {
        items =new Vector<T>(10);
    }
    public Boolean estVide(){
        return items.size()==0;
    }
    public T empiler(T item){
        items.addElement(item);
        return item;
    }
    //...
```

Suite

```
//…
public synchronized T dépiler(){
        int len = items.size();
        T item = null;
        if (len == 0)
            throw new EmptyStackException();
        item = items.elementAt(len - 1);
        items.removeElementAt(len - 1);
        return item;
    }
}
```

Autre implémentation avec listes

```
package pile;
import java.util.LinkedList;
public class SaPile<T> extends Pile<T> {
    private LinkedList<T> items;
    public SaPile(){
        items = new LinkedList<T>();
    }
    public Boolean estVide(){
        return items.isEmpty();
    }
    public T empiler(T item){
        items.addFirst(item);
        return item;
    }
    public T dépiler(){
        return items.removeFirst();
    }
}
```

Une pile de Integer

```
public class PileInteger extends Pile<Integer>{
    private Integer[] items;
    private int top=0;
    private int max=100;
    public PileInteger(){
        items = new Integer[max];
    }
    public Integer empiler(Integer item){
        if (this.estPleine())
            throw new EmptyStackException();
        items[top++] = item;
        return item;
    }
    //…
```

Suite…

```
public synchronized Integer dépiler(){
        Integer item = null;
        if (this.estVide())
            throw new EmptyStackException();
        item = items[--top];
        return item;
    }
    public Boolean estVide(){
        return (top == 0);
    }
    public boolean estPleine(){
        return (top == max -1);
    }
    protected void finalize() throws Throwable {
        items = null; super.finalize();
    }
}
```

Comment utiliser ces classes?

- Le but est de pouvoir écrire du code utilisant la classe Pile abstraite
- Au moment de l'exécution, bien sûr, ce code s'appliquera à un objet concret (qui a une implémentation)
- Mais ce code doit s'appliquer à toute implémentation de Pile

Un main

```
package pile;
public class Main {
    public static void vider(Pile p){
        while(!p.estVide()){
            System.out.println(p.dépiler());
        }
    }
public static void main(String[] args) {
        MaPile<Integer> p1= new MaPile<Integer>();
        for(int i=0;i<10;i++)
            p1.empiler(i);
        vider(p1);
        SaPile<String> p2= new SaPile<String>();
        p2.empiler("un");
        p2.empiler("deux");
        p2.empiler("trois");
        vider(p2);
    }
}
```

E) java: quelques rappels…

- Un source avec le suffixe `.java`
- Une classe par fichier source (en principe) même nom pour la classe et le fichier source (sans le suffixe `.java`)
- Méthode

```
public static void main(String[]);
```

  - `main` est le point d'entrée
- Compilation génère un `.class`
- Exécution en lançant la machine java

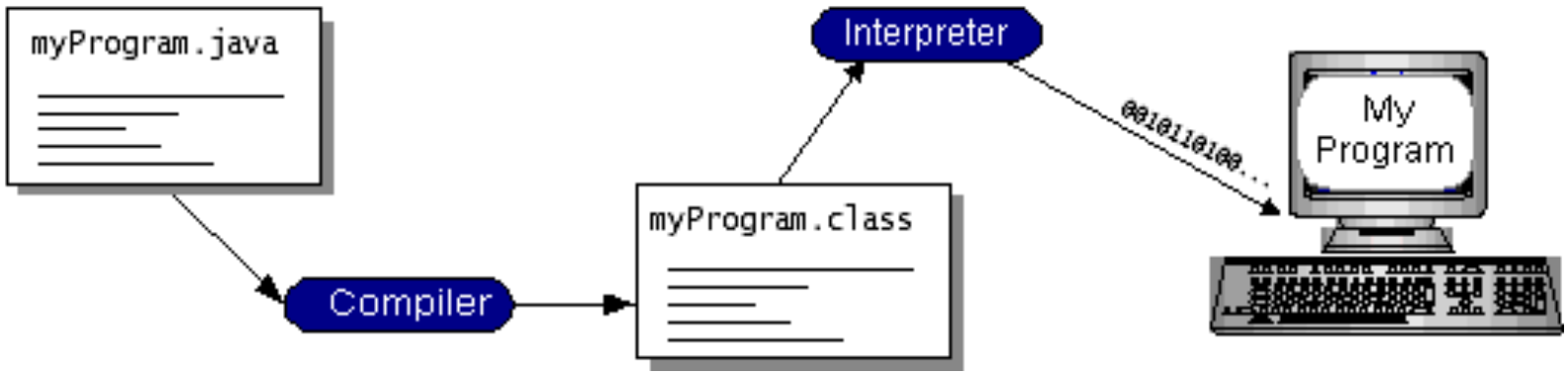
Généralités…

- Un peu plus qu'un langage de programmation:
  - "gratuit"!
  - Indépendant de la plateforme
  - **Langage interprété et byte code**
    - Portable
  - Syntaxe à la C
  - Orienté objet (classes héritage)
    - Nombreuses bibliothèques
  - Pas de pointeurs! (ou que des pointeurs!)
    - Ramasse-miettes
  - Multi-thread
  - Distribué (WEB) applet, servlet etc…
  - url: http://java.sun.com
    - http://java.sun.com/docs/books/tutorial/index.html

Plateforme Java

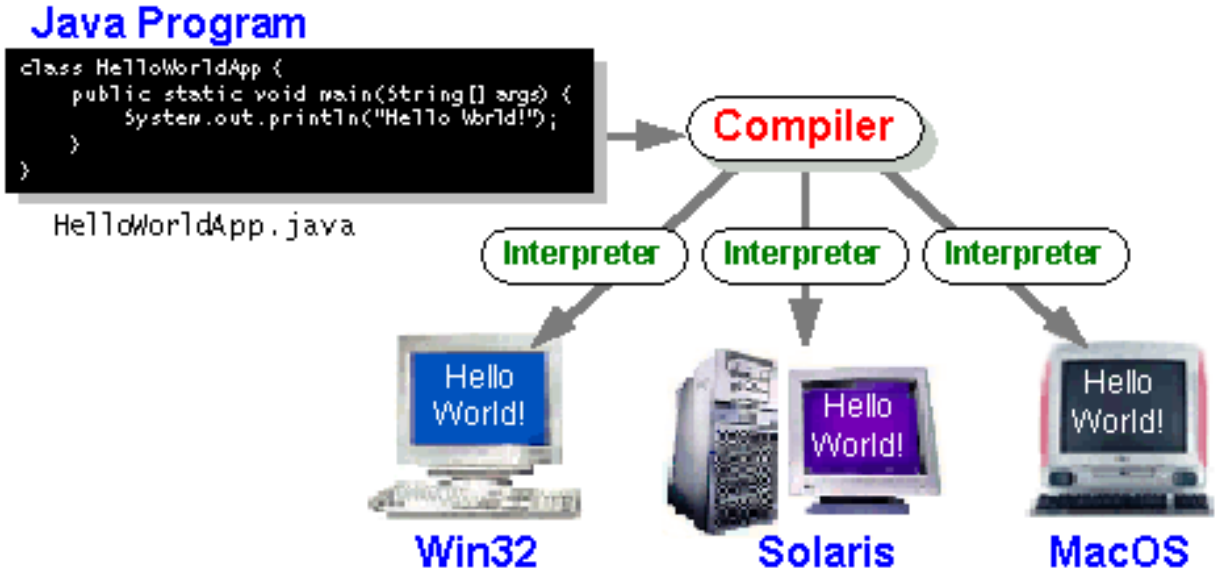
- La compilation génère un .class en bytecode (langage intermédiaire indépendant de la plateforme).
- Le bytecode est interprété par un interpréteur Java JVM

Compilation `javac`
interprétation `java`



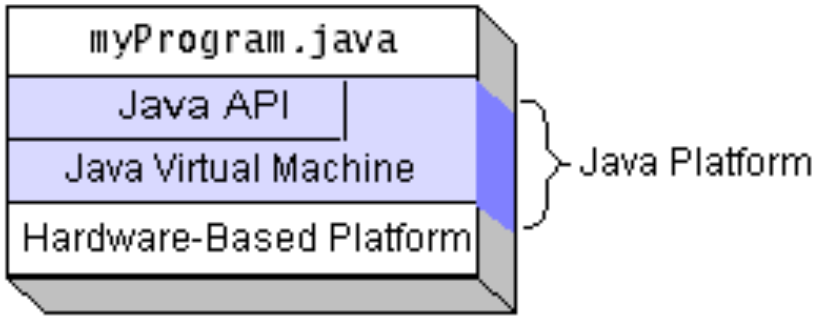
Langage intermédiaire et Interpréteur…

- Avantage: indépendance de la plateforme
  - Échange de byte-code (applet)
- Inconvénient: efficacité



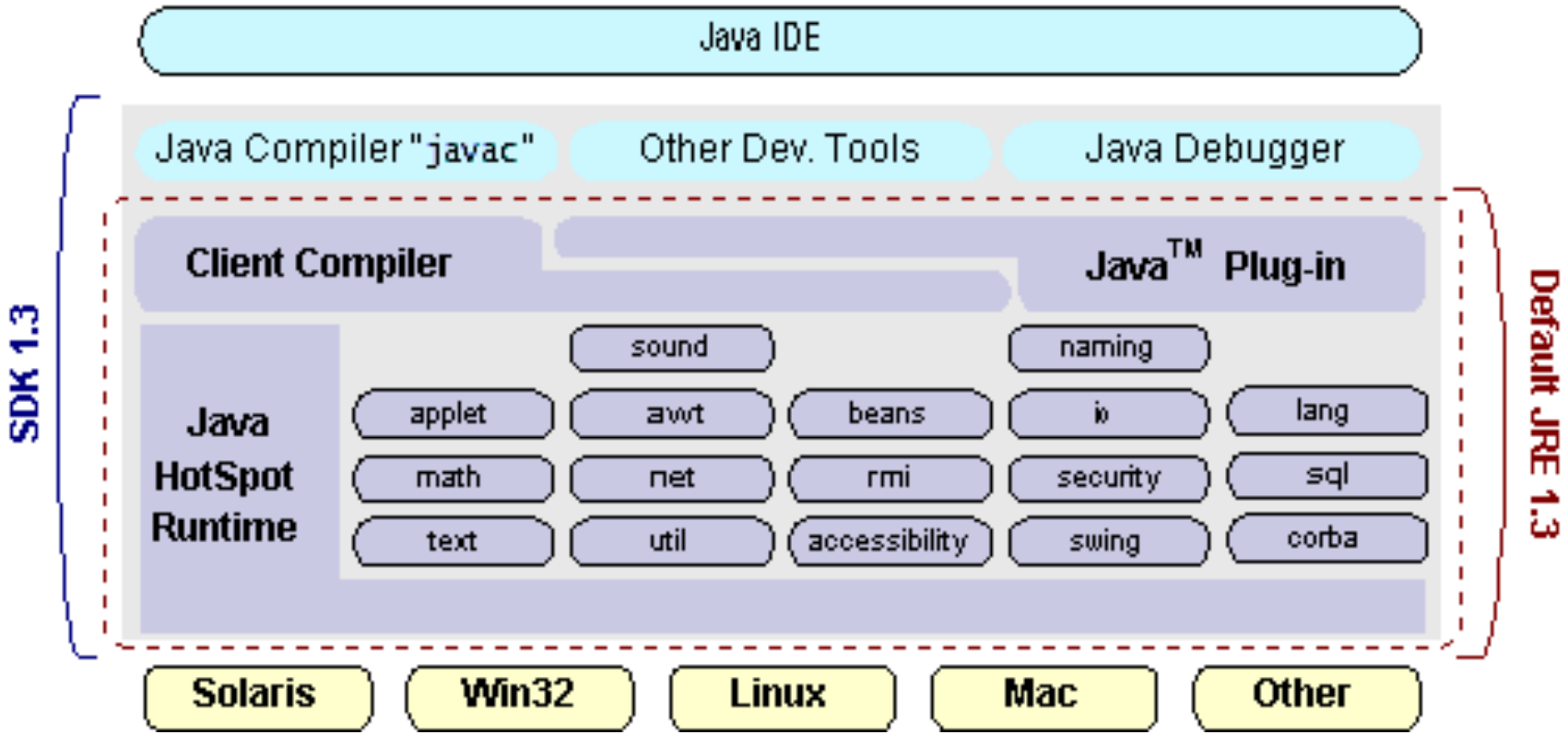
Plateforme Java

- La plateforme java: software au-dessus d'une plateforme exécutable sur un hardware (exemple MacOs, linux …)
- Java VM
- Java application Programming Interface (Java API):

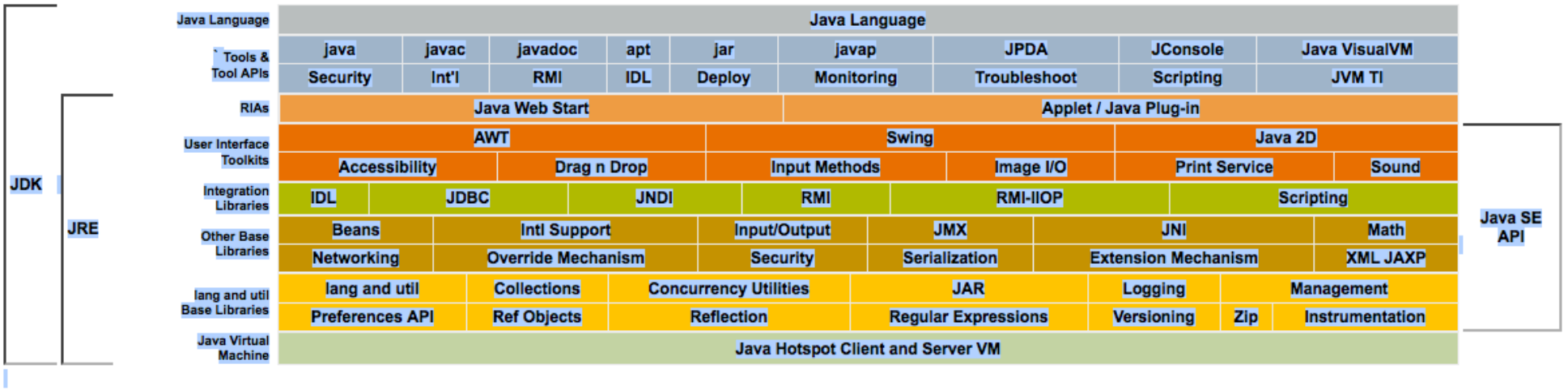


Tout un environnement…

- Java 2 sdk: JRE (java runtime environment + outils de développements compilateur, debogueurs etc…)



Tout un environnement…



Java SE 6 API Documentation

Trois exemples de base

- Une application
- Une applet
- Une application avec interface graphique

Application:

- Fichier Appli.java:

```
/**
 * Une application  basique...
 */
class Appli {
    public static void main(String[] args) {
     System.out.println("Bienvenue en L3...");
                //affichage
    }
}
```

Compiler, exécuter…

- Créer un fichier `Appli.java`
- Compilation:
  - `javac Appli.java`
- Création de `Appli.class` (bytecode)
- Interpréter le byte code:
  - `java Appli`
- Attention aux suffixes!!!
  - (il faut que javac et java soient dans $PATH)

```
Exception in thread "main" java.lang.NoClassDefFoundError:
```

  - Il ne trouve pas le main -> vérifier le nom!
  - Variable CLASSPATH ou option -classpath

Remarques

- Commentaires /* … */ et //
- Définition de classe
  - une classe contient des méthodes (=fonctions) et des variables
  - Pas de fonctions ou de variables globales (uniquement dans des classes ou des instances)
- Méthode `main`:
  - `public static void main(String[] arg)`
    - `public`
    - `static`
    - `Void`
    - `String`
  - Point d'entrée

Remarques

- Classe System
  - out est une variable de la classe System
  - println méthode de System.out
  - out est une variable de classe qui fait référence à une instance de la classe PrintStream qui implémente un flot de sortie.
    - Cette instance a une méthode println

Remarques…

- Classe: définit des méthodes et des variables (déclaration)
- Instance d'une classe (objet)
  - Méthode de classe: fonction associée à (toute la) classe.
  - Méthode d'instance: fonction associée à une instance particulière.
  - Variable de classe: associée à une classe (globale et partagée par toutes les instances)
  - Variable d'instance: associée à un objet (instancié)
- Patience…

Applet:

- Applet et WEB
  - Client (navigateur) et serveur WEB
  - Le client fait des requêtes html, le serveur répond par des pages html
  - Applet:
    - Le serveur répond par une page contenant des applets
    - Applet: byte code
    - Code exécuté par le client
    - Permet de faire des animations avec interfaces graphiques sur le client.
    - Une des causes du succès de java.

Exemple applet

- Fichier MonApplet.java:

```
/**
 * Une applet  basique...
 */
import java.applet.Applet;
import java.awt.Graphics;
public class MonApplet extends Applet {
    public void paint(Graphics g){
   g.drawString("Bienvenue en en L3...", 50,25);
    }
}
```

Remarques:

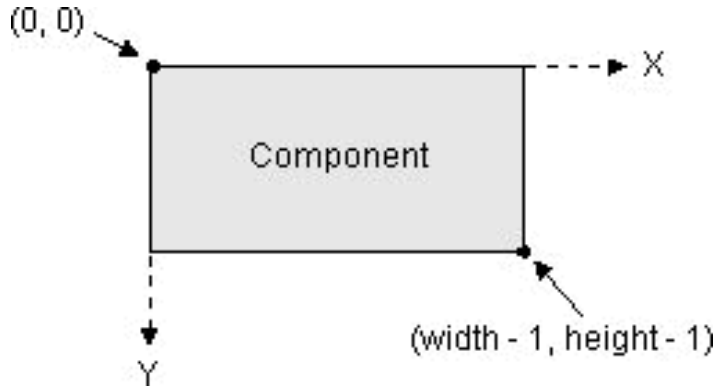
- import et package:
  - Un package est un regroupement de classes.
  - Toute classe est dans un package
  - Package par défaut (sans nom)
  - classpath
- import java.applet.*;
  - Importe le package java.applet
    - Applet est une classe de ce package,
    - Sans importation il faudrait java.applet.Applet

Remarques:

- La classe Applet contient ce qu'il faut pour écrire une applet
- … extends Applet:
  - La classe définie est une extension de la classe Applet:
    - Elle contient tout ce que contient la classe Applet
    - (et peut redéfinir certaines méthodes (paint))
  - Patience!!

Remarques…

- Une Applet contient les méthodes `paint` `start` et `init`. En redéfinissant `paint`, l'applet une fois lancée exécutera ce code redéfini.
- `Graphics g` argument de `paint` est un objet qui représente le contexte graphique de l'applet.
  - `drawString` est une méthode (d'instance) qui affiche une chaîne,
  - 50, 25: affichage à partir de la position (x,y) à partir du point (0,0) coin en haut à gauche de l'applet.



Pour exécuter l'applet

- L'applet doit être exécutée dans un navigateur capable d'interpréter du bytecode correspondant à des applet.
- Il faut créer un fichier HTML pour le navigateur.

Html pour l'applet

- Fichier Bienvenu.html:

```
<HTML>
<HEAD>
<TITLE> Une petite applet </TITLE>
<BODY>
<APPLET CODE='MonApplet.class' WIDTH=200
   Height=50>
</APPLET>
</BODY>
</HTML>
```

Html

- Structure avec balises:
- Exemples:
  - <HTML> </HTML>
  - url:
    - page de hf
- Ici:

```
<APPLET CODE='MonApplet.class' WIDTH=200
   Height=50>
</APPLET>
```

Exemple interface graphique

Fichier MonSwing.java:

```
/**
 * Une application  basique... avec interface graphique
 */
import javax.swing.*;
public class MonSwing {
    private static void creerFrame() {
        //Une formule magique...
        JFrame.setDefaultLookAndFeelDecorated(true);
        //Creation d'une Frame
        JFrame frame = new JFrame("MonSwing");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
         //Afficher un message
        JLabel label = new JLabel("Bienvenue en L3...");
        frame.getContentPane().add(label);
        //Afficher la fenêtre
        frame.pack();
        frame.setVisible(true);
    }
 public static void main(String[] args) {
     creerFrame();
    }
}
```

Remarques

- Importation de packages
- Définition d'un conteneur top-level JFrame, implémenté comme instance de la classe JFrame
- Affichage de ce conteneur
- Définition d'un composant JLabel, implémenté comme instance de JLabel
- Ajout du composant JLabel dans la JFrame
- Définition du comportement de la Jframe sur un click du bouton de fremeture
- Une méthode main qui crée la JFrame